Software Testing as a Social Science

Cem Kaner, J.D., Ph.D.

Presentation at

TASSQ

October 2006

Course materials (video lectures, etc):

www.satisfice.com/moodle

www.testingeducation.org/BBST

Source Materials on the Disk

Video lectures	Activities
Some readings	Assignments
Some instructional support material	Sample exam questions

- These materials echo sites that James Bach, Scott Barber, Tim Coulter, Rebecca Fiedler and I have been creating at Florida Tech (www.testingeducation.org) and Satisfice (www.satisfice.com) that give access to reusable content and will host supervised courses. (Some of the Satisfice-site courses will cost money, others will be free.)
- All my instructional materials are available, royalty-free, under the Creative Commons license.
- Readings on-disk that are not authored by James or me are copyrighted by others and we cannot grant Creative Commons (or other distribution) rights in them to you.

We govern our work by analogy

- *What is software testing like? Who should we look to as our role models?*
  - *Engineers?*
    - *Mechanical?*
    - *Electrical?*
  - *Project managers?*
  - *Process advocates / enforcers?*
  - *Police? (enforce "the rules")*
  - *Manufacturing-quality assurance managers?*
  - *Craftspeople?*
  - *Artists?*

Today's Assertion

Much of the most significant testing work looks more like applied psychology, economics, business management (etc.) than like programming

Social Science?

- *Social sciences study humans, especially humans in society.*
  - *What will the impact of X be on people?*
  - *Work with qualitative & quantitative research methods.*
  - *High tolerance for ambiguity, partial answers, situationally specific results.*
  - *Ethics / values issues are relevant.*
  - *Diversity of values / interpretations is normal.*
  - *Observer bias is an accepted fact of life and is managed explicitly in well-designed research.*

What's a Computer Program?

- *This year, I'm teaching intro programming.*
- *Texts define a "computer program" like this:*

A program is a set of instructions for a computer

Computer Program

A set of instructions for a computer?

- *What about what the program is for?*

- *We* **could** *define a house as a collection of construction materials assembled according to house-design patterns.*

- *But I'd* **rather** *define it as something built for people to live in.*

Computer Program

A set of instructions for a computer?

- Intent
- Stakeholders

Ignore these and lay groundwork for the classic problems of software engineering ... in the first week of the first programming class.

A different definition

- *A computer program is*
  - *a communication*
  - *among several humans and computers*
  - *who are distributed over space and time,*
  - *that contains instructions that can be executed by a computer.*

- The point of the program is to provide value to the stakeholders.

Stakeholder

- *A person*
  - *who is affected by*
  - *the success or failure of a project*
  - *or the actions or inactions of a product*
  - *or the effects of a service.*

Stakeholders (Examples)

- *Users*
- *Programmers whose code interacts with this code*
- *Maintenance programmers*
- *Technical writers*
- *Trainers*
- *Tech support staff*
- *Translators*
- *Marketers*
- *Investors*

Stakeholder

- *Favored*
  - *The people we want to satisfy*
- *Disfavored*
  - *Example: an embezzler is a stakeholder in a corporate finance system*

Stakeholder

To know how to test something, you must understand who the stakeholders are and how they can be affected by the product or system under test.

Quality is
value
to some person

– Jerry Weinberg

The Straw Man

Define system level software testing as functional, focusing on verification of the program's features, preferably against an authoritative specification.

- *Easy to understand.*
- *Easy to translate into low-skill work and routine automation.*
- *Good sales feature for academics and consultants who want to distinguish their services from software testing.*
- *NOT what you want as your career path.*

Ye Olde Straw Man

Define system level software testing as

- functional,
- focusing on verification of the program's features,
- preferably against an authoritative specification.

Traditional Focus of Testing:
Find Software Errors

But an error:

- *May or may not be a coding error*
- *May or may not be a functional error*

The tester who looks only for coding errors misses all of the other ways in which the program is of lower quality than it should be.

(This was accepted by most good testing practitioners that I knew as far back as 1983.)

Software Error

A bug is something that bugs somebody.

James Bach

Specifications & Requirements

- *What about the idea that a bug is a deviation from a specification or a failure to meet a requirement?*
  - *These MIGHT be bugs, but even if they are,*
- *Specs and requirements docs are*
  - *Probably incomplete*
    - *Individual attributes may be incompletely considered or specified*
    - *The set of attributes may be (always is) incomplete*
  - *May be infeasible*
  - *May be incompatible with other requirements or specifications*
  - *May vary across stakeholders*
  - *May change over time*
  - *May not be authoritative*

Software Error

An attribute of a software product

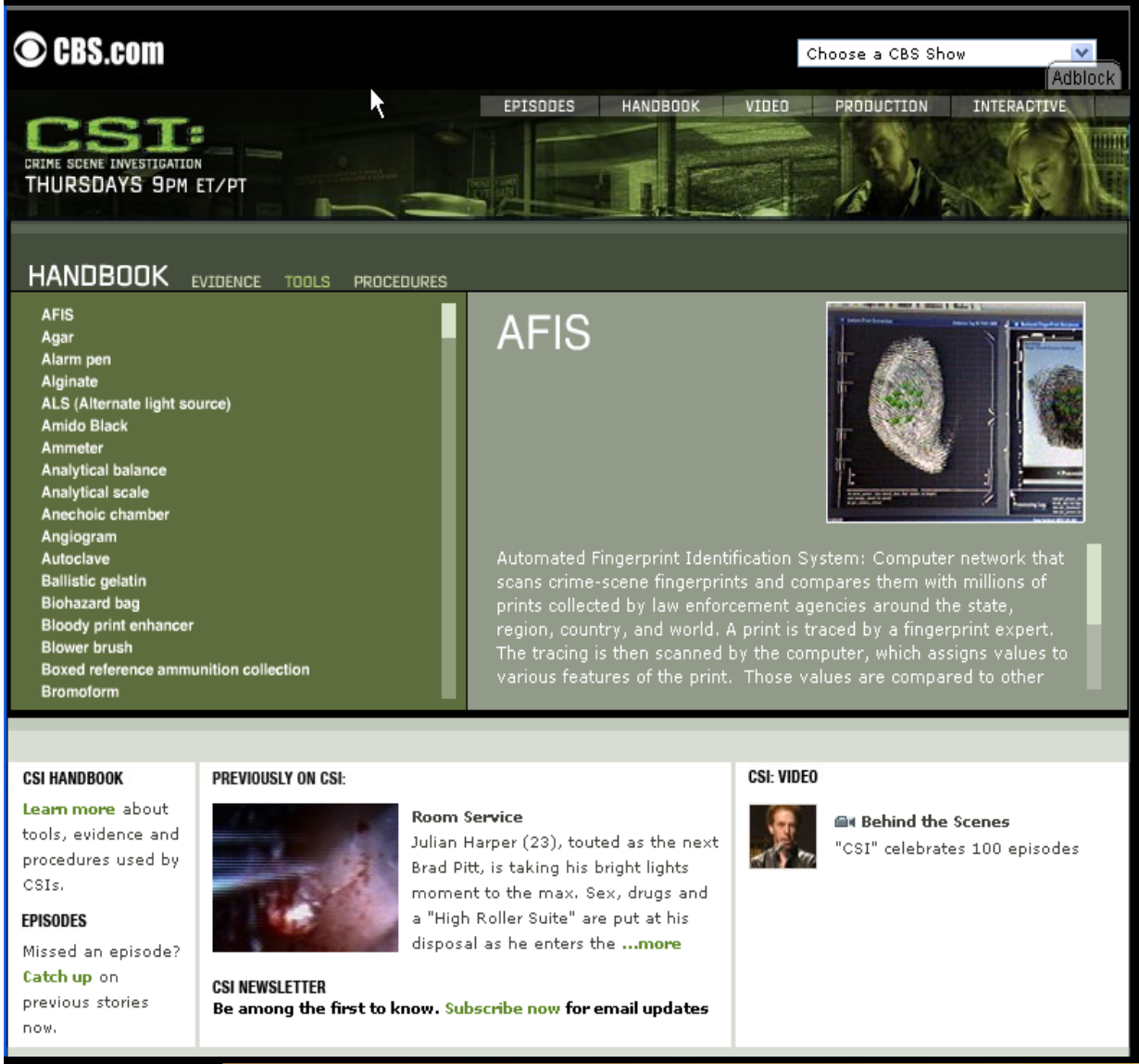
- *that reduces its value to a favored stakeholder*
- *or increases its value to a disfavored stakeholder*
- *without a sufficiently large countervailing benefit.*

Software Testing

- *A technical investigation*
- *conducted to provide quality-related information*
- *about a software product*
- *to a stakeholder.*

We used empirical methods to learn about quality

It's kind of like CSI



- MANY tools, procedures, sources of evidence.
- Tools and procedures don't define an investigation or its goals.
- There is too much evidence to test, tools are often expensive, so investigators must exercise judgment.
- The investigator must pick what to study, and how, in order to reveal the most needed information.

Information Objectives

- *Find important bugs, to get them fixed*
- *Assess the quality of the product*
- *Help managers make release decisions*
- *Block premature product releases*
- *Help predict and control costs of product support*
- *Check interoperability with other products*
- *Find safe scenarios for use of the product*
- *Assess conformance to specifications*
- *Certify the product meets a particular standard*
- *Ensure the testing process meets accountability standards*
- *Minimize the risk of safety-related lawsuits*
- *Help clients improve product quality & testability*
- *Help clients improve their processes*
- *Evaluate the product for a third party*

Different objectives drive you toward different:

- Testing techniques
- Testing-project management styles
- Results reporting methods
- Politics on the project

Test Techniques

Essentially, a test technique is a recipe for generating tests

Test Techniques

- *Examples of test techniques:*
  - *Domain testing*
  - *Function testing*
  - *Risk-based testing*
  - *Scenario testing*
  - *Transaction-flow or state-model-based testing*
  - *User testing*

Test Techniques

- *There might be as many as 150 named techniques*
- *Different techniques are useful to different degrees in different contexts*

Contexts Vary Across Projects

Testers must learn, for each new product:

- *What are the goals and quality criteria for the project*
- *What skills and resources are available to the project*
- *What is in the product*
- *How it could fail*
- *What the consequences of potential failures could be*
- *Who might care about which consequence of what failure*
- *How to trigger a fault that generates the failure we're seeking*
- *How to recognize failure*
- *How to decide what result variables to pay attention to*
- *How to decide what* other *result variables to pay attention to in the event of intermittent failure*
- *How to troubleshoot and simplify a failure, so as to better*
  - *(a) motivate a stakeholder who might advocate for a fix*
  - *(b) enable a fixer to identify and stomp the bug more quickly*
- *How to expose, and who to expose to, undelivered benefits, unsatisfied implications, traps, and missed opportunities.*

Sample Technique: Scenario Testing

The ideal **scenario** *has several characteristics:*

- *The test is* based on a story *about how the program is used, including information about the motivations of the people involved.*
- *The story is* motivating. *A stakeholder with influence would push to fix a program that failed this test.*
- *The story is* credible. *It not only* could *happen in the real world; stakeholders would believe that something like it probably* will *happen.*
- *The story involves a* complex use *of the program* or a complex environment or a complex set of data.
- *The test results are* easy to evaluate. *This is valuable for all tests, but is especially important for scenarios because they are complex.*

Note how different this is from the use-case-based scenario.

Use cases abstract out the human issues, such as motivation, and focus on sequence instead.

See John Carroll's work on scenario-based design.

16 Vectors for Creating Scenarios

- Write life histories for objects in the system. How was the object created, what happens to it, how is it used or modified, what does it interact with, when is it destroyed or discarded?
- List possible users, analyze their interests and objectives.
- Consider disfavored users: how do they want to abuse your system?
- List system events. How does the system handle them?
- List special events. What accommodations does the system make for these?
- List benefits and create end-to-end tasks to check them.
- Look at specific transactions that people try to complete, such as opening a bank account or sending a message. List all the steps, data items, outputs, displays, etc.?
- What forms do the users work with? Work with them (read, write, modify, etc.)
- Interview users about famous challenges and failures of the old system.
- Work alongside users to see how they work and what they do.
- Read about what systems like this are supposed to do. Play with competing systems.
- Study complaints about the predecessor to this system or its competitors.
- Create a mock business. Treat it as real and process its data.
- Try converting real-life data from a competing or predecessor application.
- Look at the output that competing applications can create. How would you create these reports / objects / whatever in your application?
- Look for sequences: People (or the system) typically do task X in an order. What are the most common orders (sequences) of subtasks in achieving X?

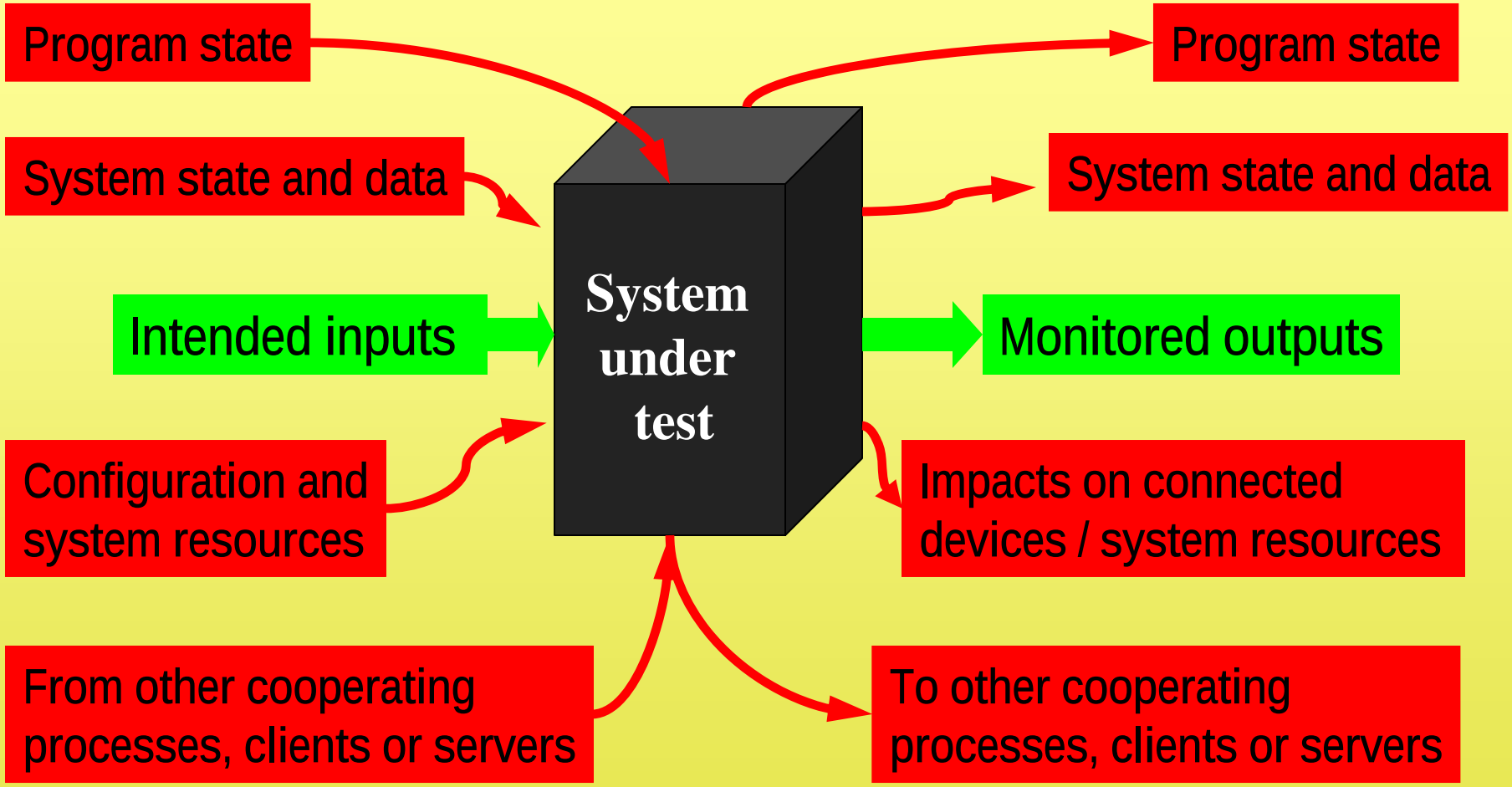
Discovering Failure is Challenging

- *Programmers find and fix most of their own bugs*
- *What testers find are what programmers missed.*
- *Testers are looking for the bugs that hide in programmers' blind spots.*
- *To test effectively, our theories of error have to be theories about the mistakes people make and when / why they make them.*

Recognizing Failure is Challenging

- *The phenomenon of inattentional blindness*
  - *humans (often) don't see what they don't pay attention to*
  - *programs (always) don't see what they haven't been told to pay attention to*
- *This is often the cause of irreproducible failures. We paid attention to the wrong conditions.*
  - *But we can't pay attention to all the conditions*
- *The 1100 embedded diagnostics*
  - *Even if we coded checks for each of these, the side effects (data, resources, and timing) would provide us a new context for the Heisenberg principle*
- *Our Tests Cannot Practically Address All of the Possibilities*

Recognizing Failure is Challenging



Based on notes from Doug Hoffman

And *Even If* We Demonstrate a Failure That Doesn't Mean Anyone Will Fix It

- *The decision to fix a bug is rooted in a cost / benefit analysis*
- *The quality of the bug description (the effectiveness of the tester) lies in its:*
  - *Technical quality (scope and severity)*
  - *Impact analysis (what are costs to who)*
  - *Persuasiveness and clarity*

A Few More Assertions

- *Most testing metrics are human performance metrics*
  - *How productive is this tester?*
  - *How good is her work?*
  - *How good is someone else's work?*
  - *How long is this work taking them?*

- *These are well studied questions in the social sciences and not well studied when we ignore the humans and fixate on the computer.*

Let's Sum Up

- *Is testing ONLY concerned with the human issues associated with product development and product use?*
  - *Of course not*
  - *But thinking in terms of the human issues leads us into interesting questions about*
    - *what tests we are running (and why)*
    - *what risks we are anticipating*
    - *how/why these risks are important, and*
    - *what we can do to help our clients get the information they need to manage the project, use the product, or interface with other professionals.*